Cooperative Distributed Deep Neural Network Deployment with Edge Computing

Cian-You Yang[‡], Jian-Jhih Kuo[†], Jang-Ping Sheu[‡], and Ke-Jun Zheng[§]
[‡]Dept. of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
[†]Dept. of Computer Science & Information Engineering, National Chung Cheng University, Chiayi, Taiwan
[§]Dept. of Information Systems and Applications, National Tsing Hua University, Hsinchu, Taiwan
s106062584@m106.nthu.edu.tw, lajacky@cs.ccu.edu.tw, sheujp@cs.nthu.edu.tw, s108065536@m108.nthu.edu.tw

***Abstract*—Deep Neural Networks (DNNs) are widely used to analyze the abundance of data collected by massive Internet-of-Thing (IoT) devices. The traditional approaches usually send the data to the cloud and process the DNN inference on the powerful cloud servers, but suffer from long network latency. Therefore, edge computing has emerged to reduce network latency by offloading the computation from the cloud to the edge. However, a single resource-constrained edge device is unable to process real-time DNN inference. Thus, we devise a collaborative edge computing system CoopAI to distribute DNN inference over several edge devices with a novel model partition technique to allow the edge devices to prefetch the required data in advance to compute the inference cooperatively in parallel without exchanging data. Subsequently, we present a new optimization problem to minimize the completion time of distributed DNN inference. An innovative algorithm is then proposed to intelligently partition the model into the proper number and sizes of blocks, deploy them on a suitable number of edge devices, and run them in different rounds. The numerical results manifest that our algorithm outperforms the traditional approach by 20%−30% on the completion time.**

***Index Terms*—Distributed deep neural network, edge computing, dynamic programming**

I. Introduction

Recently, the ever-faster-growing computing architectures have facilitated Deep Neural Networks (DNNs) to achieve impressive success in various applications [1] such as image recognition, speech recognition, and natural language processing. Typically, the Internet-of-Things (IoT) devices (e.g., IP-enabled cameras) continuously generate and upload data (e.g., video) to the cloud, and the cloud employs DNNs to extract accurate information from data streams of IoT devices to make an inference with powerful servers [2]. However, the IoT devices are increasing sharply and thus generate a considerable surge of data.[i] The traditional paradigm is pushed to the limit and has the following issues. First, time-sensitive applications like virtual/augmented reality (VR/AR) and smart traffic light, may suffer from long transmission latency since the cloud is often far from the IoT devices. Second, uploading a large amount of data may overwhelm the backhaul network due to the explosive increase in IoT devices and applications [4]. Last, uploading data to the cloud may cause privacy issues [5].

To remedy these issues, we employ the following two concepts: 1) Edge computing [4]: Edge computing has emerged to offload the computation from the cloud to the edge to take services closer to users. 2) Model partition and layer slicing [6]–[10]: The idea is to compute the DNN model with multiple devices in parallel to accelerate the inference. With the concepts, a novel system termed CoopAI is then presented to speed up the inference of DNNs. In CoopAI, the IoT devices monitor the targets and *continuously* transmit data to gateway for inference, and the gateway coordinates the edge devices to cooperate in inference (detailed in Section II).

Model partition divides a model into several *blocks* and processes each block in a different *round* as shown in Fig. 1. Layer slicing aims to divide the computation in a block into *independent* tasks performed by different edge devices in parallel. Specifically, the convolutional layer (CL) in a block is sliced into *grids*, each of which *processes a horizontal slice of an input image independently* as shown in Fig. 2 (i.e., data parallelism), whereas the fully-connected layer (FL) in a block is sliced into several *clusters*, each of which *generates a part of or a subset of output independently* as shown in Figs. 3 and 4 (i.e., data or model parallelism).[ii] The gateway collects intermediate results of each round from edge devices and then relays the results to the edge devices for computing the next block. However, traditional systems limit *the blocks to only one layer each* (i.e., *layer-by-layer partition*) [6]–[10] for simplicity, which may cause frequent data exchange between the edge devices and significantly slow down the inference.

Thereby, our system CoopAI innovates *multi-layer partition*, which allows multiple layers to be grouped into a *block* (see Section II) to *process multiple layers in a round*. Edge devices that work together on multiple layers yet require intermediate results from each other, which was achieved by frequent data exchange and layer-by-layer partition in traditional systems. To address the issue, CoopAI permits each edge device to *prefetch* extra data to *compute* the required intermediate results by itself for *avoiding data exchange during a round* with multiple layers (detailed in Section II-A). However, *model partition* for the proper number and sizes of blocks (i.e., rounds) and the influence of *edge device number* on the completion time (i.e., computing and transmission time) have not been investigated before. In addition, *layer slicing* for multiple layers in a block has not been studied in the literature to ensure that edge devices do not exchange data with each other during a round. Hence, each grid (or cluster) sliced from layers can be executed independently. Thus, the *time-aware partition for DNNs via edge computing* that jointly decides model partition and layer slicing has not been explored to partition the model

[i]International Data Corporation predicts that IoT devices grow to 41.6 billion and generate 79.4 zettabytes of data [3] by 2025.

[ii]Section III of [11] presents more details about data and model parallelism.

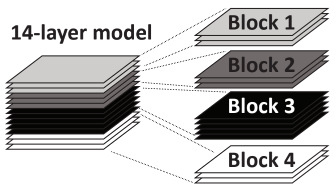


Fig. 1. A model divided into blocks.

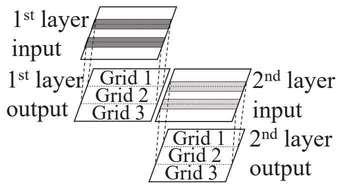


Fig. 2. Layer-by-layer CL partition.

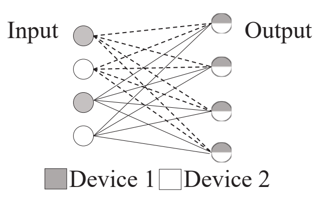


Fig. 3. Data parallelism for a FL

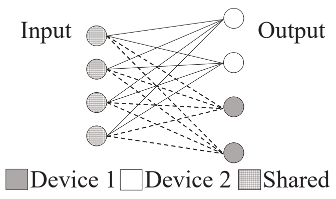


Fig. 4. Model parallelism for a FL

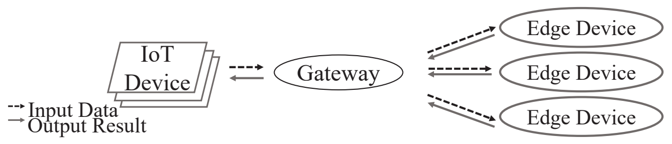


Fig. 5. Overview of CoopAI

into moderate-size blocks (i.e., rounds), slice the layers into a proper number of grids (or clusters), and deploy the grids (or clusters) on the edge devices to minimize the completion time.

In this paper, therefore, we make the first attempt to explore the time-aware partition for DNNs via edge computing. Simultaneously optimizing model partition and layer slicing raises three new challenges as follows. 1) *Elastic model partition*: Partitioning a model into fewer blocks (i.e., fewer rounds) tends to reduce the number of communications between the gateway and devices. However, the edge devices have to compute more prefetched data for a larger block during a round to acquire the intermediate data by themselves. The extra computing time may cause overlength completion time and deteriorate user experience of time-sensitive applications. 2) *A Suitable number of devices*: Intuitively, the more involved edge devices may enjoy the more powerful total computing capability but share the limited bandwidth of the gateway, which may induce longer transmission time and further prolong the completion time of distributed DNN inference. 3) *The Best mode selection*: A DNN model typically consists of CLs and FLs.[iii] Compared to FLs, CLs are more likely to be grouped into a block to reduce the number of rounds (i.e., avoid frequent communications). The reason is that data prefetching makes each edge device compute all required intermediate result by itself (see Section II-A), and then edge devices can compute grids independently in a round. By contrast, it is difficult to have more than two FLs in a block because deriving a result of a neuron in a FL requires all the results of the neurons in the preceding layer (see Section II-B). Therefore, time-aware partition for DNNs via edge computing is quite challenging since it has to jointly determine whether, where, and how to compute model partition and layer slicing.

To address these challenges, we first formulate a new optimization problem termed **Co**operative **D**NN Deployment via **E**dge Computing (CODE). With the given parameters: 1) a DNN model with layer information and 2) the number of candidate edge devices, CODE aims to 1) partition the model, 2) slice the layers, and 3) coordinate the edge devices such that the completion time for inference is minimized. We then study the different modes of layer slicing, derive four feasible cases of blocks, and acquire the recursive recurrence for CODE. Subsequently, a dynamic programming algorithm termed **M**ulti-**L**ayer Partition and **S**licing (MLS) is proposed based on the four feasible cases of blocks to address *suitable number of devices*, *elastic model partition*, and *best mode selection*. Specifically, MLS first examines every possible block with different consecutive layers and records their completion time (including computing and transmission time). Then, with the recursive recurrence, the solution of every sub-problem can be obtained by combining the solution of a smaller sub-problem with one additional block. Finally, we conduct extensive simulations with the data of real platforms and well-known DNNs to verify the performance of MLS.

[iii]This paper mainly considers the CLs and FLs since they are the most resource-consuming part of DNN models compared with the other layers such as pooling layers [6], [12].

II. Novel System Model - CoopAI

The system CoopAI is an edge computing environment, which consists of IoT devices, a gateway, and edge devices. The overview of CoopAI is depicted in Fig. 5. To enable the *time-sensitive* applications (e.g., real-time image recognition applications) such as VR/AR gaming experience, elderly care, smart traffic light, the IoT devices (e.g., IP-enabled cameras) *continuously* transmits the data (e.g., images) to the gateway for inference by adopting a pre-trained DNN model. We assume that the DNN model is stored in each device and does not require frequent updates. Thus, the download time of the DNN model from the gateway to the edge devices can be ignored.

The gateway decomposes the inference task and coordinates the nearby devices to cooperate in inference. To speed up the inference, it partitions the model into several *blocks* and slices the CLs (or FLs) into multiple *girds* (or *clusters*). The grids (or clusters) in a block should be computed by the devices *in parallel*, and each device is only allowed to communicate with the other devices via the gateway at the beginning and the end of a round. In addition, each edge device only requests the missing data (i.e., not the output computed by itself) required in the next round. However, different from the traditional systems, CoopAI innovates *multi-layer partition* rather than employs *layer-by-layer partition*. Thus, each block may include a subset of *consecutive* layers, each grid processes a horizontal slice of an input image for CLs, and each cluster generates part of output or a subset of output for FLs. In the following, we introduce *advanced* slicing for *multiple* layers in CoopAI.

A. Layer slicing for multiple CLs

Fig. 6 shows layer slicing for *multiple* CLs. According to [11], CoopAI aims to slice multiple CLs in the same block into grids with *data parallelism* such that the grids can be computed by different edge devices in parallel. Similar to slicing a single layer in Fig. 2, each gird in a layer requires the output data of the neighboring grids in the preceding layer to be

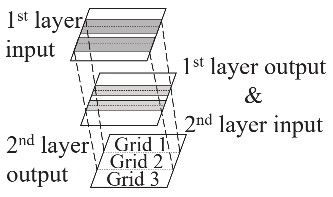


Fig. 6. Multi-layer partition for CLs.

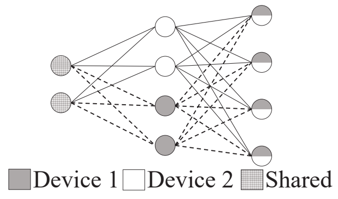


Fig. 7. Hybrid model for two FLs.

its input data in each round. Therefore, slicing multiple layers is more complicated since each device requires some intermediate results from the other devices for computing succeeding layers but edge devices cannot exchange data during a round (i.e., compute independently). For example, in Fig. 2, grid 2 in the 2^{nd} layer inevitably requires the output data of grid 1 and grid 3 in the 1^{st} layer (i.e., the light gray areas of 2^{nd} layer input).

To resolve the problem, CoopAI innovates the notion, *data prefetching*, which allows an edge device to prefetch the additional input data in the other grids at the beginning of a round such that the input data of later layers can be computed by itself. For example, the device processing grid 2 will prefetch the additional input data in advance as shown in the dark gray areas of 1^{st} layer input in Fig. 6. Thus, the edge devices, each of which process a different grid, can compute the output in the block by themselves in parallel without data exchange during a run. Nevertheless, there is a trade-off between *layer-by-layer partition* and *multi-layer partition* for the model partition of CLs. Layer-by-layer partition has more communication rounds (i.e., longer transmission time) but has less computing overhead (i.e., shorter computing time) compared to multi-layer partition, which leads to the challenge, *elastic model partition*.

B. Layer slicing for multiple FLs

Compared with CLs, it is difficult to group more than two FLs into a block, because deriving the output of a neuron in a FL requires all the output of the neurons in the preceding layer. Moreover, slicing two *consecutive* FLs by leveraging only data parallelism or only model parallelism is *unrealistic* since they cannot avoid data exchange during a round. However, two FLs in a block can be sliced by the *hybrid mode*, i.e., model parallelism on the 1^{st} FL and data parallelism on the 2^{nd} FL.

Specifically, the output of a cluster in the 1^{st} layer sliced by model parallelism contains only a subset of the original output data, whereas the output of a cluster is exactly the input of the cluster in the 2^{nd} layer sliced by data parallelism. In other words, each edge device can compute its corresponding cluster in the two layers without data exchange with the other devices. Fig. 7 depicts an example of two FLs grouped into a block and the two layers in a block are sliced into two clusters being computed by two devices in parallel. Finally, the output of the 2^{nd} layer will be summed by the gateway to derive the final output. Thus, CoopAI groups *at most two FLs into a block*.

III. Problem Formulation

We formulate the problem as an optimization problem termed **Co**operative **D**NN Deployment via **E**dge Computing (CODE) for our system CoopAI as follows. In CODE, the edge computing environment consists of $D \in \mathbb{Z}^+$ *identical*[iv] edge devices (e.g., Raspberry Pi 3/4) with the computing capability $F \in \mathbb{R}^+$ (i.e., floating-point operations per second) and a gateway with the bandwidth capacity $G \in \mathbb{R}^+$ (i.e., Mbps). The IoT devices (e.g., cameras) monitor the targets and transmit the data (e.g., images or video) to the gateway for inference. The data is analyzed by a given L-layer DNN model that typically consists of CLs and FLs, where $L \in \mathbb{Z}^+$. Note that each CL j has Ci_j input channels and Co_j output channels with the filter size $\mathcal{F}_r$. The height and width of each CL j are denoted by H_j and W_j, whereas the input and output size of each FL j are denoted by Ni_j and No_j. CoopAI has to answer the following questions to minimize the completion time (including the computing and transmission time).

First, CODE has to partition the DNN model into B moderate-size blocks and each block is allowed to have a different number of layers, where $B \in \mathbb{Z}^+ \cap [1, L]$. In other words, the inference is partitioned into B rounds and each round should process a moderate number of layers. Second, CODE asks for the suitable number k of edge devices to process *different* grids or clusters sliced from the layers in the block for each round in parallel, where $k \in \mathbb{Z}^+ \cap [1, D]$. Note that the more involved edge devices have the more total computing capability (i.e., $k \cdot F$), while sharing the limited bandwidth capacity (i.e., $\frac{G}{k}$ for each selected edge device). Last, CODE aims to select the best mode for slicing multiple layers in every block. The modes of any two consecutive rounds (i.e., data parallelism, model parallelism, or hybrid mode introduced in Section II) should be jointly examined to select those that balance the computing and transmission time to minimize the completion time of all rounds.

The following constraints should be carefully addressed. Each grid or cluster sliced from the layers in a block should be executed by an edge device without data exchange with the other devices during the round. That is, the edge devices are *only allowed to communicate* with each other via the gateway *at the beginning and end of each round*. In addition, for each device, *it suffices* to communicate with the other devices for the *missing* data (i.e., not the output computed by itself) that is required in the next round. Finally, each grid or cluster should have a *similar computing overhead* to avoid overwhelming one of edge devices while idling the other edge devices.

IV. Algorithm Design

We propose an algorithm termed **M**ulti-**L**ayer Partition and **S**licing (MLS) to carefully address all the challenges of CODE. MLS exploits *dynamic programming* by first computing and recording the optimal solution of each smaller sub-problem in CODE, and then reusing these solutions to iteratively solve a larger sub-problem. For each sub-problem, MLS obtains the optimal solution by effectively combining 1) each one related to the previous solution that groups a specific number of layers in preceding rounds with 2) one extra round that groups the remaining layers to explore different combinations effectively.

[iv]To explore the intrinsic property of CODE, it is reasonable to assume that the computing capability of edge devices in the system is *identical* since the edge computing platforms are usually built by users (e.g., Raspberry Pi 3/4).

That is, MLS carefully examines every possible combination to decide the number of blocks (i.e., rounds) and the sizes of blocks (i.e., *elastic model partition*) to reduce the number of communications while maintaining an acceptable prefetched data size. Moreover, all possible numbers of edge devices are examined to find the *suitable number of devices*. The recursive recurrence for MLS is introduced in Section IV-A. Besides, to achieve *the best mode selection*, MLS inspects the slicing mode of every layer in each round (i.e., data/model parallelism or hybrid mode) to optimize the computing and transmission time in Section IV-B. Due to the page limit, more details, examples, and pseudocode are presented in the technical report [11].

A. Recursive Recurrence

Specifically, let T_i^k denote the minimum completion time for processing the first i layers (i.e., from layer 1 to layer i) of the DNN model with k devices. Thereby, the minimum completion time for processing the entire DNN model (i.e., L layers) with at most D devices (denoted by T_L^*) is

$$T_L^* = \min_{\forall k \in [1,D]} T_L^k. \tag{1}$$

It then suffices to derive T_i^k, where $1 \leq i \leq L$ (i.e., every *sub-problem* of T_L^k). Generally speaking, it can be envisaged that the solution of T_i^k must deploy a specific number of consecutive layers (i.e., from layer 1 to layer r) in the preceding blocks and deploy the rest of layers (i.e., from layer $r+1$ to layer i) in an additional succeeding block. However, if the number of edge devices is one (i.e., $k=1$), then only 1) the input data transmission for the first layer from the gateway to the edge device and 2) the output data transmission for the last layer from the edge device to the gateway are required. It is because all the input data of layers 2 to L are computed and output by the edge device itself. Therefore, MLS especially employs the traditional approach to calculate the completion time for the case of $k=1$. To calculate the completion time of the sub-problem, let $b_{i,j}^k$ denote the block including layers i to j of the DNN model with k devices, and $e_{i,j}^k$ denote the minimum completion time of $b_{i,j}^k$, where $1 \leq i \leq j \leq L$. Therefore, T_i^k is derived by the following recursive recurrence.

$$T_i^k = \begin{cases} 0, & \text{if } i = 0; \\ T_{i-1}^k + e_{i,i}^k, & \text{else if } k = 1; \\ \min_{0 \leq r < i}(T_r^k + e_{r+1,i}^k), & \text{otherwise.} \end{cases} \tag{2}$$

Note that the *base case* (i.e., if $i=0$) returns zero since no layer is processed. Before introducing how to derive $e_{i,j}^k$ in Section IV-B, we analyze the time complexity of MLS and give an example to illustrate the above steps in MLS as follows.

Time Complexity. Once all variables $e_{i,j}^k$ are found in advance, where $1 \leq i \leq j \leq L$ and $1 \leq k \leq D$, the time complexity for computing the dynamic-programming-based algorithm via eq. (1) is $O(DL^2)$. ■

B. Minimum Completion Time for Computing a Block

The completion time for computing a block includes the computing time and transmission time, and thus MLS derives the minimum completion time $e_{i,j}^k$ for computing block $b_{i,j}^k$ in eq. (2) by eq. (3). Let $\zeta_{i,j}^k(d)$ denote the number of floating-point operations[v] (FLOPs) for computing the grid (or cluster) sliced from block $b_{i,j}^k$ assigned to device d, and let $\varphi_{i,j}^k(d)$ denote the data size transmitted between the gateway and device d for computing block $b_{i,j}^k$. After that, we have

$$e_{i,j}^k = \max_{\forall d \in [1,k]} \frac{\zeta_{i,j}^k(d)}{F} + \frac{\varphi_{i,j}^k(d)}{\frac{G}{k}}. \tag{3}$$

Recall that each edge device has *identical* computing capability F, whereas the bandwidth G of the gateway will be shared by the selected edge devices. Thereafter, each edge device has the bandwidth capacity $\frac{G}{k}$. For ease of reading, the computing overhead $\zeta_{i,j}^k(d)$ and transmission overhead $\varphi_{i,j}^k(d)$ are derived later in Sections IV-B1 and IV-B2.

By eq. (3), MLS finds the *best mode selection* for each block, and then derives the minimum completion time for k edge devices to compute any block that consists of layers i to j (i.e., $e_{i,j}^k$), where $1 \leq k \leq D$ and $1 \leq i \leq j \leq L$. Meanwhile, since the blocks partitioned from our model may not be sliced into equally-sized grids (or clusters), the edge devices need to wait for the slowest edge device among them to start the next round. Moreover, by recursive recurrence in eq. (2), MLS then iteratively solves a sub-problem by reusing the recorded solutions of smaller sub-problems to achieve the *elastic model partition* and derive the solution of T_L^k. Finally, by eq. (1), MLS determines the *suitable number of devices* for computing the DNN cooperatively with multiple edge devices.

1) Computing Overhead for Computing a Block: To further calculate $\zeta_{i,j}^k(d)$, MLS sums up the computing overhead of each layer r (denoted by $p_{i,j,r}^k(d)$) in the block $b_{i,j}^k$, where $i \leq r \leq j$. Therefore,

$$\zeta_{i,j}^k(d) = \sum_{r \in [i,j]} p_{i,j,r}^k(d). \tag{4}$$

MLS calculates $p_{i,j,r}^k(d)$ based on [14]. Recall that layer r could be a CL or FL, and thereby MLS considers the two types of layers separately. If layer r is a CL, the computing overhead $p_{i,j,r}^k(d)$ is proportional to the number of times that edge device d executes the filters of layer r. Otherwise, $p_{i,j,r}^k(d)$ is proportional to the input size times the output size executed on edge device d based on the selected mode (data parallelism, model parallelism, or hybrid mode). Due to the page limit, the derivation of $p_{i,j,r}^k(d)$ is presented in [11].

2) Transmission Overhead for Computing a Block: The size of data transmission between the gateway and device d for block $b_{i,j}^k$ (i.e., $\varphi_{i,j}^k(d)$) includes 1) the *input* data size from the gateway to the edge devices and 2) the *output* data size from the edge devices to the gateway. However, compared to the input data size, MLS has to handle two *non-trivial* different states for the output data size as follows.

i) The output data size of a block depends on the number of layers in the succeeding block if *the succeeding block has CLs* since the succeeding block that contains the more CLs

[v]FLOPs is widely used to evaluate the computing overhead of inference for DNN models [13].

TABLE I
INFORMATION OF ADOPTED DNN MODELS

	AlexNet	VGG16	VGG19	YOLOv2
Input size	224*224	224*224	224*224	608*608
Parameters (million)	62	138	144	51
Operations (GFLOPs)	2.27	31.0	39.3	62.9

needs to prefetch the more data for parallel computing (see Section II-A).

ii) Otherwise, the output data size of the block can be determined according to the last layer in the block.

To this end, MLS subtly calculates the output data size for each block *at a different timing* according to the state of the block. For the first state, the calculation of output data size for the block is postponed to the time when MLS processes the succeeding block. For the second state, MLS immediately calculates the output data size for the block according to the last layer in the current block. Thus, three following variables are introduced to acquire $\varphi_{i,j}^{k}(d)$.

- Variable $ti_{i,j}^{k}(d)$ denotes the input data size of layer i transmitted from the gateway to device d.
- Variable $\tau o_{i,j}^{k}(d)$ denotes the output data size of layer $i-1$ transmitted from device d to the gateway *if layer i is a CL*; otherwise, it is zero.
- Variable $to_{i,j}^{k}(d)$ denotes the output data size of layer j transmitted from device d to the gateway *if layer $j+1$ is a FL*; otherwise, it is zero.

Note that the variables $ti_{i,j}^{k}(d)$ and $to_{i,j}^{k}(d)$ denote the input and output data size of the *current* block, respectively. By contrast, the variable $\tau o_{i,j}^{k}(d)$ represents the output data size of the *preceding* block. Thus,

$$\varphi_{i,j}^{k}(d) = ti_{i,j}^{k}(d) + \tau o_{i,j}^{k}(d) + to_{i,j}^{k}(d). \tag{5}$$

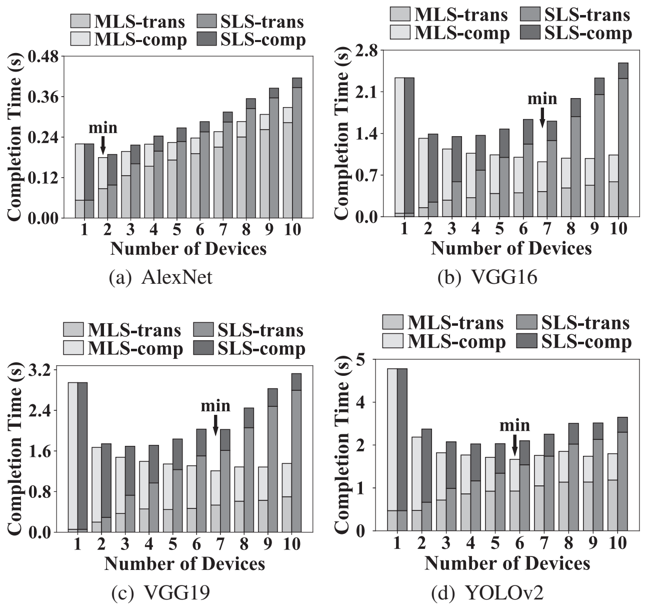
To obtain $\varphi_{i,j}^{k}(d)$, the variables, $ti_{i,j}^{k}(d)$, $\tau o_{i,j}^{k}(d)$, and $to_{i,j}^{k}(d)$ are derived with considering all possible sub-cases. Due to the page limit, the derivation of the variables is presented in [11].

V. EVALUATION

We first verify and validate all proposed formulas and the good results are presented in [11]. Then, the performance of our algorithm MLS is evaluated by extensive simulations based on the data of real platforms and well-known DNN models. The two adopted platforms for the gateway and edge devices are 1) Raspberry Pi 3 Model B (RP3) with quad-core 1.2 GHz ARM Cortex-A53 processor with 1 GB RAM[vi] and 2) Raspberry Pi 4 (RP4) with quad-core 1.5GHz ARM Cortex-A72 processor with 4GB RAM. The information of the adopted DNN models, AlexNet [15], VGG16 [16], VGG19 [16], and YOLOv2 [17], are briefly summarized in Table I. AlexNet is the most lightweight DNN model among the adopted models, whereas YOLOv2 is the most computation-intensive one.

We assume that the selected edge devices evenly share the bandwidth of the gateway via TCP/IP. MLS is compared with the layer-by-layer partition and slicing (SLS) algorithm based on [6], [7]. Recall that all pre-trained DNNs are stored on the devices beforehand and the download time for the model

[vi]The results of RP3 are presented in [11] due to the page limit.



(a) AlexNet (b) VGG16
(c) VGG19 (d) YOLOv2

Fig. 8. Effect of number of selected devices on completion time using RP4 for different DNN models

TABLE II
NUMBER OF BLOCKS PARTITIONED BY MLS ON RP4

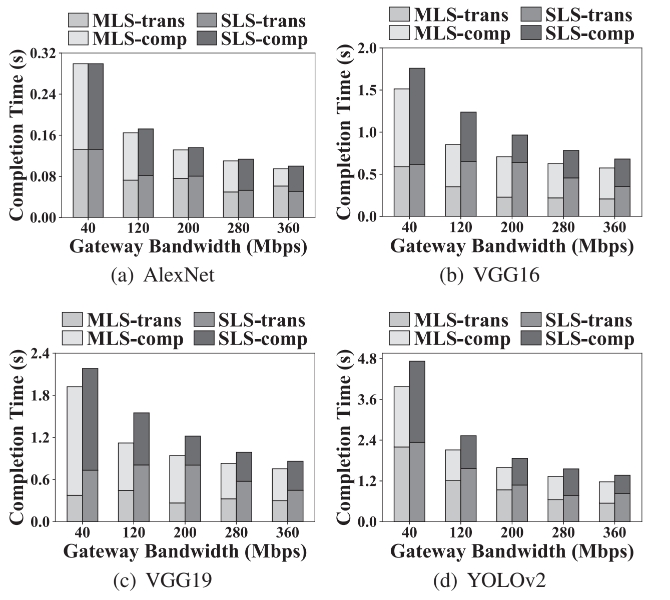
# Devices	1	2	3	4	5	6	7	8	9	10
AlexNet	1	5	4	4	4	4	4	4	4	4
VGG16	1	8	7	6	5	4	4	4	4	4
VGG19	1	11	9	8	5	4	5	5	4	4
YOLOv2	1	12	10	9	8	7	7	7	5	5

is ignored. The default gateway bandwidth and computation capability of edge devices for the RP4 cases are set to 100 Mbps and 13.6 GFLOPS. The number of edge devices are changed from one to ten to evaluate the effect of the number of devices on 1) completion time, 2) computing overhead, and 3) transmission overhead. Then, we also change the gateway bandwidth from 40 to 400 Mbps to evaluate the effect of gateway bandwidth on the completion time for the RP4 cases.

A. Effect of Number of Edge Devices RP4

Generally speaking, Fig. 8 shows that MLS outperforms SLS on completion time on RP4. For RP4, 2, 7, 7, and 6 devices are chosen by MLS to save completion time by 4.91%, 31.49%, 28.53%, 17.75% for the four models compared to the best cases of SLS (i.e., 2, 3, 3, and 4 devices). MLS can further reduce completion time up to 18.43%, 60.46%, 58.94%, and 55.91% for the four models compared to using only one device. It is because MLS carefully decides the number and sizes of blocks partitioned from the model (i.e., *elastic model partition*) and selects the *suitable number of devices* to balance the computing and transmission time with eq. (2). Besides, MLS always examines the three possible modes (i.e., data parallelism, model parallelism, and hybrid mode) to determine the *best mode selection* (i.e., *four feasible cases* in [11]) for slicing each block to further reduce the completion time.

More specifically, Fig. 8 also shows the computing and transmission overhead per edge device. The computing overhead of MLS is *slightly* more than that of SLS and leads to a longer computing time. However, MLS can reduce more



(a) AlexNet (b) VGG16 (c) VGG19 (d) YOLOv2

Fig. 9. Effect of gateway bandwidth on completion time using RP4 for different DNN models

transmission time and has the ability to balance computing and transmission time. Thus, the overall completion time of MLS is shorter than that of SLS. Fig. 8 also indicates that a small number of edge devices are sufficient to compute AlexNet since it is very lightweight. The more devices will cause the additional transmission time, which is much more than the reduced computing time with more devices. By contrast, the other three models are more computation-intensive, and the more edge devices can further reduce the completion time.

Fig. 8 also manifests that the completion time of SLS tends to decrease as the number of devices increases (except for AlexNet). That is because the computing overhead per device decreases when the number of edge devices increases. However, the completion time slows down the reduction as the number of edge devices keeps increasing. Two following reasons increase the transmission time sharply: 1) the transmission overhead per edge device may increase as the number of edge devices increases , and 2) the gateway bandwidth is *shared* by the increasing edge devices. By contrast, MLS can lower the bad effect by expanding the block size (i.e., fewer blocks) to reduce the transmission overhead when the number of devices increases (i.e., *elastic model partition*) (see Table II).

B. Effect of Gateway Bandwidth

Fig. 9 shows the effect of gateway bandwidth on the completion time, where the number of available devices is set to ten. Therefore, MLS and SLS will select the device number for the best performance. MLS can reduce the completion time up to 5.71%, 31.10%, 27.69%, and 18.09% for the four models compared to SLS. When the gateway bandwidth increases, the performance gap between MLS and SLS becomes smaller. It is because MLS tends to put the more layers in a block and has to more prefetched data for computing a round, and thus slightly sacrifice the computing time for reducing the number of communication rounds and transmission time. However, the trade-off advantage becomes limited as the transmission time accounts for a very small proportion of the completion time. In other words, it is difficult to further decrease the completion time when the computing time dominates the completion time. It is worth noting that the completion time of MLS will never exceed that of SLS since MLS will also employ layer-by-layer partition and slicing for the DNN model in the worst case.

VI. Conclusions

In this paper, we investigate DNN inference acceleration with edge computing and innovate CoopAI to allow each block has a different number of layers (i.e., multi-layer partition and layer slicing). We present a novel optimization problem to minimize the completion time by balancing the computing and transmission time and propose a dynamic programming algorithm termed MLS to carefully address the new challenges, *elastic model partition*, *suitable number of devices*, and *best mode selection* to achieve the optimal solution. Finally, the extensive simulation results with real DNNs and real platforms manifest that MLS outperforms the traditional layer-by-layer partition and layer slicing by 20%−30% in most cases. The interplay between computing capability and gateway bandwidth is also analyzed to show how MLS to balance the computing and transmission time to reduce the completion time.

References

[1] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2224–2287, 2019.

[2] X. Xu *et al.*, "A heuristic offloading method for deep learning edge services in 5G networks," *IEEE Access*, vol. 7, pp. 67 734–67 744, 2019.

[3] C. MacGillivray and D. Reinsel, "Worldwide global DataSphere IoT device and data forecast, 2019–2023," 2019.

[4] X. Wang *et al.*, "In-edge AI: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Network*, vol. 33, no. 5, pp. 156–165, 2019.

[5] S. A. Osia, A. S. Shamsabadi, A. Taheri, H. R. Rabiee, and H. Haddadi, "Private and scalable personal data analytics using hybrid edge-to-cloud deep learning," *Computer*, vol. 51, no. 5, pp. 42–49, May 2018.

[6] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2348–2359, Nov 2018.

[7] J. Mao *et al.*, "MoDNN: Local distributed mobile computing system for deep neural network," in *Proc. DATE*, 2017.

[8] R. Hadidi *et al.*, "Musical chair: Efficient real-time recognition using collaborative iot devices," *CoRR*, vol. abs/1802.02138, 2018.

[9] R. Hadidi, J. Cao, M. Woodward, M. S. Ryoo, and H. Kim, "Distributed perception by collaborative robots," *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 3709–3716, Oct 2018.

[10] J. Mao *et al.*, "MeDNN: A distributed mobile system with enhanced partition and deployment for large-scale DNNs," in *Proc. IEEE/ACM ICCAD*, 2017.

[11] C.-Y. Yang, J.-J. Kuo, J.-P. Sheu, and K.-J. Zheng, "Cooperative distributed deep neural network deployment with edge computing," Oct 2020. [Online]. Available: http://hscc.cs.nthu.edu.tw/submit/paper.pdf

[12] A. Coates, B. Huval, T. Wang, D. J. Wu, A. Y. Ng, and B. Catanzaro, "Deep learning with COTS HPC systems," in *Proc. ICML*, 2013.

[13] A. Sehgal and N. Kehtarnavaz, "Guidelines and benchmarks for deployment of deep learning models on smartphones as real-time apps," *CoRR*, vol. abs/1901.02144, 2019.

[14] P. Molchanov *et al.*, "Pruning convolutional neural networks for resource efficient transfer learning," in *Proc. ICLR*, 2017.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. NIPS*, 2012.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. ICLR*, 2015.

[17] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," in *Proc. IEEE CVPR*, 2017.